

Affine Collapse and Conditional Composition in Linear Systems

A useful way to think about many modern computational systems is in terms of “affine collapse.” A stack of ordinary linear transformations often collapses into a single equivalent operator, causing much of the apparent internal structure to disappear.

For example, a simple two-stage linear system:

$$y = BAx$$

can always be rewritten as:

$$y = Mx$$

where:

$$M = BA$$

From the outside, the intermediate structure no longer matters. The system behaves as one affine transformation. Many different choices of A and B may produce exactly the same effective matrix M, leading to parameter redundancy and loss of independent degrees of freedom.

The interesting question is therefore:

What kinds of mechanisms prevent this collapse into one fixed operator?

A major answer is conditional participation of parameters.

Binary Switching Between Matrix Pairs

Consider a system where the input first encounters one of two matrices:

$$A_1 \text{ or } A_2$$

followed by one of another pair:

B_1 or B_2

The computation flow is:

$x \rightarrow A_i \rightarrow B_j \rightarrow y$

Since matrix multiplication composes right-to-left, the four possible effective operators are:

B_1A_1

B_1A_2

B_2A_1

B_2A_2

giving the four computations:

B_1A_1x

B_1A_2x

B_2A_1x

B_2A_2x

This arrangement also aligns naturally with binary control logic:

Control Bits	Effective Operator
00	B_1A_1
01	B_1A_2

10 B_2A_1

11 B_2A_2

The important point is that the system no longer realizes one affine map. Instead, it realizes a family of affine maps selected conditionally.

Because each matrix participates in multiple composites, the parameters are no longer easily absorbed into a single equivalent transform. Each matrix becomes a reusable transformation component within a combinatorial operator family.

With k independently switched stages, the number of possible effective operators grows exponentially as:

$$2^k$$

even though only a small number of matrices are stored.

Data-Dependent Modulation

Binary switching is only one way to prevent affine collapse. Another is data-dependent modulation.

A particularly important structure is:

$$y = AD_{\phi}Bx$$

where D_{ϕ} is a diagonal matrix whose entries depend on the input (shown by ϕ .)

Without the diagonal modulation:

$$ABx$$

collapses into one affine operator.

With modulation:

$$AD_n Bx$$

the effective operator becomes:

$$M(x) = AD_n B$$

which changes over input space.

The system is no longer globally affine because the transformation itself depends on the signal.

This introduces:

- conditional routing,
- multiplicative interactions,
- adaptive gain control,
- context-dependent computation.

ReLU Networks as Dynamically Gated Linear Systems

ReLU neural networks can be interpreted in exactly this way.

A ReLU layer:

$$h = \text{ReLU}(Wx+b)$$

can be rewritten as:

$$h = D_n(Wx+b)$$

where the diagonal entries of D_n are binary:

$$D_{ii} \in \{0,1\}$$

depending on whether the neuron activation is positive.

A two-layer ReLU network therefore has the form:

$$y = W_2 D_r W_1 x$$

This is not merely a nonlinear function. It is a dynamically gated linear system.

For different inputs, different diagonal gates are activated, producing different effective operators:

$$M_r = W_2 D_r W_1$$

for different activation regions r .

In effect, ReLU networks dynamically select among many affine sub-networks.

An inactive ReLU neuron removes:

- its own output row,
- and a corresponding input column in the next matrix.

This dynamically edits the computational graph itself.

Noncommutativity and Permutation

An even richer possibility appears when the ordering of matrices becomes data dependent.

Matrix multiplication is generally noncommutative:

$$AB \neq BA$$

so ordering carries computational meaning.

Suppose a system contains matrices:

A, B, C

Then:

ABC

and:

CBA

are generally different operators.

A controller can therefore encode computation not only through matrix selection, but through matrix ordering.

With n matrices, there are:

$n!$

possible permutations.

Even small matrix pools can therefore generate very large operator families through dynamic ordering alone.

This turns the system from a fixed transformation into a programmable composition process whose geometry depends on sequencing and interaction structure.

A Broader Interpretation

These examples suggest that much of the expressive power in modern computational systems does not come from deep stacks of affine transforms alone.

The real power appears when parameter participation becomes conditional through mechanisms such as:

- switching,
- gating,
- modulation,

- routing,
- permutation,
- noncommutative composition.

The effective operator becomes:

$M(x, s, \text{state}, t, \dots)$

Denoting an effective transformation matrix whose coefficients are not fixed, but vary according to the current input x , control signals s , internal system state, time t , or other contextual variables. Instead of implementing one static operator, the system dynamically constructs or selects an operator from a larger family of possible transformations during computation.

From this viewpoint, systems such as ReLU networks, mixture-of-experts models, adaptive transforms, and dynamically modulated Hadamard constructions can all be seen as different ways of navigating a large space of conditional operators rather than implementing one static affine map.